



COSC 2206 Internet Tools

JavaScript

Browser versions
Language Versions
Core Language



Netscape browser versions

- Netscape 2 (JavaScript 1.0, obsolete)
- Netscape 3 (JavaScript 1.1)
- Netscape 4 (JavaScript 1.2, broken)
- Netscape 4.5 (JavaScript 1.3)
- Netscape 6 / Mozilla (JavaScript 1.5)



Microsoft browser versions

- IE 3 (JScript 1.0 / 2.0)
- IE 4 (JScript 3.0)
- IE 5 (JScript 5.0)
- IE 5.5 (JScript 5.5)
- IE 6 (JScript 5.5)



Standardized Versions

- The ECMA (European Computer Manufacturers Association) has standardized JavaScript (ECMA-262) called ECMAScript
- There are 4 versions so far (v1, v2, v3, v4)
- Search web for ECMAScript if you want more information.
- [http://www.ecma-
international.org/publications/standards/Ecm
a-262.htm](http://www.ecma-international.org/publications/standards/Ecm
a-262.htm)



Language Versions (1)

- JavaScript 1.0
 - original obsolete version
- JavaScript 1.1
 - fixed bugs, introduced proper arrays, implemented in Netscape 3
- JavaScript 1.2
 - added regular expression, almost compliant with ECMAScript v1, implemented in Netscape 4



Language Versions (2)

- JavaScript 1.3
 - compliant with ECMAScript v1, implemented in Netscape 4.5
- JavaScript 1.4
 - server version
- JavaScript 1.5
 - added exception handling, compliant with ECMAScript v3, implemented in Netscape 6 and Mozilla



Language Versions (3)

- JScript 1.0
 - like JavaScript 1.0, implemented in IE 3
- JScript 2.0
 - like JavaScript 1.1, implemented in IE 3
- JScript 3.0
 - like JavaScript 1.3, compliant with ECMAScript v1, implemented in IE 4
- JScript 4.0
 - standalone non-browser version



Language Versions (4)

- JScript 5.0
 - added exception handling, almost compliant with ECMAScript v3, implemented in IE 5
- JScript 5.5
 - like JavaScript 1.5, compliant with ECMAScript v3, implemented in IE 5.5 and IE 6
- ECMAScript v1, v2
 - First standardized version, v2 (maintenance release)
- ECMAScript v3 (regex and exceptions)



Main language references (1)

- Netscape has online client guide to JavaScript:

<http://devedge.netscape.com/>



Main language references (2)

- Netscape has online JavaScript client reference at **`developer.netscape.com`**

<http://devedge.netscape.com/>



Main language references (3)

- For information on Microsofts versions of javascript called JScript search for JScript at

<http://msdn.microsoft.com>



Other references (1)

- w3schools has an excellent interactive tutorial on JavaScript

<http://www.w3schools.com>



Reference book

- JavaScript: The Definitive Guide, 4th Edition
David Flanagan
O'Reilly, 2002
ISBN: 0-596-00048-0
- This is like four books in one
 - Core JavaScript
 - Client-Side JavaScript
 - Core JavaScript Reference
 - Client-Side JavaScript Reference



What is JavaScript

- High-level scripting (interpreted) language
- Untyped, prototype based OOP language
- Not a simple language
- Used to be called LiveScript and has no connection with Java
- Borrows a lot of syntax from Java and C
- It can run in a browser (client-side) or as a standalone scripting language (Microsoft's JScript and WSH)

Where do we put JavaScript?

- Embedded in an HTML document between script tags

```
<script language="javascript">
```

JavaScript statements go here

```
</script>
```

- In an external file which is loaded using

```
<script src="program.js" ...></script>
```



JavaScript file



Nothing between tags



Where do we put `<script>`?

- **In the head of the HTML document**

- Here it is read before the HTML document in the body is parsed. Any code, except function definitions, will be executed immediately.

- **In the body of the HTML document**

- Here it is read while the HTML document is being parsed. When the parser sees the `<script>` tag it stops parsing the document and interprets the JavaScript code.



Writing HTML with JavaScript


- Later we will discuss the document object model (DOM) that lets JavaScript interact with the elements of an HTML document.
- For now we will use `document.write()` to produce HTML output using JavaScript.
- Example:

```
document.write("<h1>Hello World</h1>");
```



Hello World program (1)

```
<html>
<head>
<title> ... </title>
<script language="javascript">
    document.writeln("<h1>Hello World</h1>");
</script>
</head>
<body>
</body>
</html>
```



Anything here will display
after the Hello World

<examples/simple/HelloWorld1.html>



Hello World program (2)

```
<html>
<head>
<title> ... </title>
</head>
<body>
<script language="javascript">
    document.writeln("<h1>Hello World</h1>");
</script>
</body>
</html>
```

[examples/simple/HelloWorld2.html](#)



Hello World program (3)

```
<html><head><title> ... </title>
<script language="javascript">
function hello()
{ document.write("<h1>Hello World</h1>"); }
</script>
</head>
<body>
<script language="javascript">hello();</script>
</body>
</html>
```



<examples/simple/HelloWorld3.html>



Hello World program (4)

```
<html><head><title> ... </title>
<script language="javascript" src="hello.js">
</script>
</head>
<body>
<script language="javascript">hello();</script>
</body>
</html>
```

<examples/simple/HelloWorld4.html>

```
function hello()
{
    document.writeln("<h1>Hello World</h1>");
}
```



hello.js

Hello World program (5)

```
<html><head><title> ... </title>
<script language="javascript">
    window.alert("Hello World");
</script>
</head>
<body>
Close alert box to see this text.<br>
Use reload to run the script again.
</body>
</html>
```



An alert box

<examples/simple/HelloWorld5.html>

Hello World program (6)

```
<html><head><title> ... </title>
<script language="javascript">
    function hello()
    { document.write("<h1>Hello World</h1>"); }
</script>
</head>
<body onload="hello()">
This text will never be seen.
</body>
</html>
```

if you do this the
back button won't
work

this is
Microserf
specific

<examples/simple/HelloWorld6.html>



Insert date example

```
<html><head><title> ... </title>
```

```
</head>
```

```
<body>
```

```
<h1>Inserting the date into a document</h1>
```

The date is

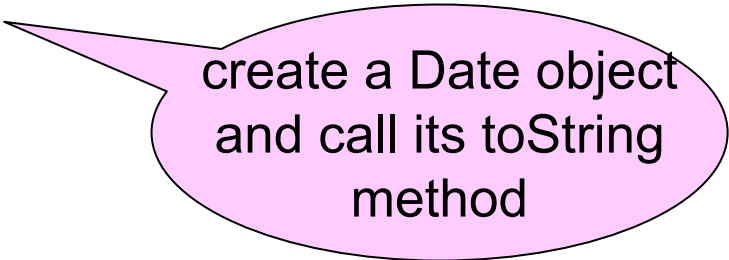
```
<script language="javascript">
```

```
    document.write(new Date());
```

```
</script> right now.
```

```
</body>
```

```
</html>
```



create a Date object
and call its toString
method

<examples/simple/insertDate.html>

JavaScript Variables

- JavaScript is an untyped language so a variable can hold any kind of value.
- The **var** keyword is used to define variables.

```
var i = 1;
```

```
i = "a string";
```

```
i = new Date();
```

No type is used

- If it is omitted the variable is implicitly declared as a variable with global scope.
- **JavaScript is case-sensitive**



Variable types

- Just because JavaScript is untyped doesn't mean that there are no data types:
- Number type
 - no distinction between integer and floating point numbers (64 bits)
- String type and string literals
 - literals enclosed in single or double quotes
 - strings are like immutable objects
- Array type, Object type, many other types

Variable Scoping (1)

- Unlike Java, JavaScript is not block-scoped
- Example:

```
{  
    var sum = 0.0;  
    ...  
}
```

In Java, sum is
local to the block.

In JavaScript
it is local to the
enclosing function



Variable scoping (2)

- A local variable (declared inside a function) will hide a global variable with the same name.
- Attempting to use a variable that has been declared but not initialized gives the **undefined** value. In this sense all declared variables have a value.



Number Data Type

- Number is an object type that represents integers and floating point numbers
- Floating point numbers are 64-bit IEEE
- There is no separate integer type
- A floating point value can store an integer exactly in the range -2^{53} to 2^{53}
- Floating point literals have a decimal point or **e**, **E** to represent the exponent.
- **+**, **-**, *****, **/** are floating point operations



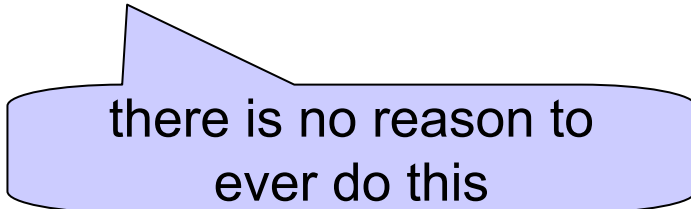
Special Number values

- **Infinity**
- **NaN**
 - not a number
- **Number.MAX_VALUE**
 - largest positive number
- **Number.MIN_VALUE**
 - smallest non-zero positive number
- **Number.POSITIVE_INFINITY**
- **Number.NEGATIVE_INFINITY**



String Data Type (1)

- Strings are like immutable objects and are very similar to strings in Java
- String literals can use single or double quotes as delimiters. `\` acts as an escape so `\n` refers to the newline character.
- The following are the same
 - `var s = "hello";`
 - `var s = new String("hello");`



there is no reason to
ever do this



String Data Type (2)

- There are many string methods and most are the same as in Java
- `s.length` (this is a property)
- `s.substring(i)`
- `s.substring(i, j)`
- `s.charAt(i)`
- `s.indexOf(pattern)`



Boolean data type

- Like Java, the literals **true** and **false** are the values of a boolean variable in JavaScript
- Unlike Java, JavaScript can convert them to 1 and 0 to use in expressions as needed



Standard functions

- Functions are very different than in Java
- We can have standard function definitions like

```
function square(x)
{
    return x * x;
}
```

- No types are specified.
- Function like this are similar to Java methods



Function literals (1)

- They have no correspondence in Java
- Function literals are unnamed functions like the lambda functions in Lisp that can be assigned as the value of a variable.
- Example

```
var square =  
    function(x) {return x*x;};
```

- Used the same way as a standard function:

```
var s = square(2.0);
```



Function Literals (2)

- Function literals can be used as arguments to other functions:

```
var comp = function compare(x,y)
    { return x - y; }
```

- Now if we have a **sort** function that needs a comparison function we can call it like

```
a.sort(comp)
```

where **a** is the array to sort



The Function constructor

- Functions can be constructed at run-time using a function constructor:

```
var square = new Function  
    ("x", "return x*x;");
```

- Here the first argument is the function argument list and the second argument is the function body.
- There is no correspondence in Java



Object Types (1)

- An uninitialized object

```
var obj = new Object();
```

- A Date object for today

```
var today = new Date();
```

- A Date object for Jan 1, 2002, 0 hours

```
var newYear =  
    new Date(2002, 0, 1, 0, 0, 0, 0);
```



Object Types (2)

- Data fields of a Java object are properties in JavaScript. They are always public and are accessed directly using the dot notation:

`document.myForm.myName`

- Here `document` is a predefined object referring to the HTML document, `myForm` is the name of a form and `myName` is the name of the specified input object. For example

`<input type="text" name="myName" ...>`



Object Types (3)

- If a Rectangle object `rect` has `width` and `height` properties they can be accessed as
`rect.width` (same as `rect['width']`)
`rect.height` (same as `rect['height']`)
- Methods are also invoked using dot notation
- Instance method example
`document.write(...)` ;
- Static method example
`var s2 = Math.sqrt(2.0)` ;



Object Types (4)

- Simple custom objects (no methods)

```
var point = new Object();
```

```
point.x = 3.2;
```

```
point.y = -1.7;
```

- point is an object with properties **x** and **y**
- Object literals can also be used to directly construct objects

```
var point = { x:3.2, y:-1.7 };
```



Array Types (1)

- Unlike Java, arrays are dynamic in Java
- Example: Declare an array with no elements and then add some elements

```
var a = new Array()
```

```
a[0]=1; a[1]=2; a[2]=3; a[9]=10;
```

- There are 10 elements but 6 are undefined
- Length of array is `a.length`

<examples/simple/array1.html>



Array Types (2)

- Arrays can be inhomogeneous:

```
var b = new Array(123, "Fred",  
    345.50, new Array(1,2,3));
```

- Array literals can be used

```
var b = [123, "Fred", 345.50];  
var m = [ [1,2,3], [4,5,6], [7,8,9] ];
```

- There are many array methods (later)

<examples/simple/array2.html>



Array Types (3)

- Arrays can be associative:

```
var age = new Array();  
age['fred'] = 34; age['jim'] = 13;  
age['bob'] = 27;  
for (var name in age)  
{ document.write("Age of " + name +  
  " is " + age[name] + "<br>");  
}
```

<examples/simple/array3.html>



Primitive & Reference Types

- Numbers and booleans are primitive types
- Call by value is used for primitive types
- Objects are reference types as in Java
- Call by value is also used for reference types but the value is a reference as in Java
- Arrays and Strings are also reference types
- Strings are immutable. They have references but act like primitive types since the reference cannot be used to change them.



The null value

- As in Java, object references can be assigned the value `null` to indicate that they don't yet refer to any object.



Comments

- Two styles of comments

- `/* */`

- `//`



JavaScript Operators (1)

- For the most part Java and JavaScript operators are similar
- There are some important differences.
- JavaScript has `=`, `==`, and `===` operators.
 - `=` is used for assignment
 - `==` is used to test for equality
 - `===` is used to test for identity



JavaScript Operators (2)

- Numbers, strings and boolean values are compared by value so `==` and `===` have the same meaning.
- For strings in Java `==` is useless since it compares references but in JavaScript it compares the characters in the strings.
- Objects are compared by reference so `==` compares references.



JavaScript Operators (3)

- In general, for objects the rules for `==` and `===` are complicated.
- If you are defining your own objects then write your own equality method.
- For Strings the operators `<=`, `<`, `>`, `>=`, `==`, and `!=` compare characters (in Java we need `compareTo`). In JavaScript there is also a `localeCompare` () method.



JavaScript Operators (4)

- The **in** and **instanceof** operators
- The **in** operator can be used to check if an object has a certain property.
- It can also be used to iterate over the properties of an object.
- The **instanceof** operator is same as in Java and is used to check if an object has a given type.



JavaScript Operators (5)

- String Concatenation operator
 - use +, numbers are converted to strings if one operand is a number and the other is a string.
- There are many string methods too
 - see later



JavaScript Operators (6)

- The **typeof** operator
- Unary operator to check the generic data type of a variable.
- The possible values are "number", "string", "boolean", "object", "undefined"
- Use instanceof to distinguish among different object types.



JavaScript Operators (7)

- the **delete** operator
- Unary operator
- Can be used to delete a property of an object
- Can be used to delete an array element
- Note: JavaScript has garbage collection, like Java, so delete is not related to this kind of deletion.



JavaScript Operators (8)

- The **new** operator
- Unary operator
- As in Java it is used to construct an object.



JavaScript Operators (9)

- the **void** operator
- A strange unary operator
- When it is applied to a method call expression it throws away the return value and returns the undefined value.
- Used in hypertext links:

```
<a href="javascript:void ..." >...</a>
```

so that browser won't display return value



Statements (1)

- Expression Statements

- Examples:

- `count++;`
- `alert("This is a warning");`
- `document.write("Hello");`



Statements (2)

- Assignment statements

- Examples

- `var y = 1.0;`

- `x = 1.5;`

- `hypot = Math.sqrt(x*x + y*y);`

- `w = window.open(...);`



Statements (3)

- Compound statement

```
{
```

```
    one or more statements
```

```
}
```



Statements (4)

- **if**, **if-else**, and **if-else if** statements
- These have the same structure as the corresponding statement in Java. Example:

```
if (...)
{
    . . .
}
else
{
    . . .
}
```



Statements (5)

- **while** statement
- Same structure as the corresponding statement in Java. Example:

```
while (...)
{
    ...
}
```



Statements (6)

- **do-while** statement
- Same structure as the corresponding statement in Java. Example:

```
do
{
    . . .
} while ( . . . ) ;
```



Statements (7)

- **for** statement
- Same structure as the corresponding statement in Java (more like the C++ for loop than Java)

```
for ( . . . ; . . . ; . . . )  
{  
    . . .  
}
```



Statements (8)

- **for in** statement

```
for(var property in object)
{
    ...
}
```

- The following example shows how to display a table of window and document properties

<examples/simple/for-in.html>



Statements (9)

- The **return** statement returns a value from a function
- Example:

```
return 1.0 - Math.pow(1.0 + i, -n) / i;
```



Statements (10)

- Other types of statements
 - `var`
 - `label`
 - `continue`
 - `function`
 - `throw`
 - `try / catch / finally`
 - `with`



factorial function (1)

```
// Compute n!  
function factorial(n)  
{  
    var p = 1;  
    for (var k = 2; k <= n; k++)  
        p = p * k;  
    return p;  
}
```



factorial function (2)

```
function factorialTable()  
{  
    document.write("<pre>");  
    for (var k = 0; k <= 25; k++)  
    {  
        document.writeln(k + "! = " +  
            factorial(k));  
    }  
    document.write("</pre>");  
}
```

<examples/simple/factorial.html>



reverse string function (1)

```
// reverse the string s
```

```
function reverse(s)
```

```
{
```

```
    var sReverse = "";
```

```
    for (var k = 0; k < s.length; k++)
```

```
    {
```

```
        sReverse = s.charAt(k) + sReverse;
```

```
    }
```

```
    return sReverse;
```

```
}
```



reverse string function (2)

```
// Recursive version
```

```
function recursiveReverse(s)
{
    if (s.length <= 1) return s;
    return
        recursiveReverse(s.substring(1)) +
            s.charAt(0);
}
```

<examples/simple/reverse.html>



max function (1)

// finding max value in an array

```
function maxArray(a)
{
    var maxValue = Number.NEGATIVE_INFINITY;
    for (var k = 0; k < a.length; k++)
    {
        if (a[k] > maxValue) maxValue = a[k];
    }
    return maxValue;
}
```



max function (2)

- It can be tested using the HTML

The maximum of the numbers 3,4,5,-1,-2 is

```
<script language="javascript">  
    document.write(maxArray([3,4,5,-1,-2]));  
</script>
```



A literal array



max function (3)

// finding max value in arg list

function **max**()

Math.max works like this

{

var maxValue = **Number.NEGATIVE_INFINITY**;

for (var k = 0; k < arguments.**length**; k++)

{

if (arguments[k] > maxValue)

maxValue = arguments[k];

}

return maxValue;

}

special array of
function arguments



max function (4)

- It can be tested using the HTML

The maximum of the numbers 3,4,5,-1,-2 is

```
<script language="javascript">  
    document.write(max(3,4,5,-1,-2));  
</script>
```

<examples/simple/max.html>



format function (1)

```
function format(n, w)
{
    var val = Math.round(n * 100) / 100;
    var s = val + "";
    if (s.indexOf(".") < 0) s = s + ".00";
    if (s.indexOf(".") == s.length-2) s = s + "0";
    var spaces = w - s.length;
    for (var k = 1; k <= spaces; k++)
    {
        s = " " + s;
    }
    return s;
}
```



format function (2)

- format right justifies the number **n** in a field of width **w** characters. It can be tested using the HTML

The rounded value of 34.999999999 is

```
<script language="javascript">  
    document.write(format(34.999999999,1)) ;  
</script>
```

[examples/simple/format.html](#)



Array methods (1)

- Arrays are created using
 - Example: `var a = new Array(...);`
- The argument, if any is the number of array elements to allocate initially.
- Arrays can also be created using an array literal
 - Example:
`var a = ["abc", 2, true, [1,2,3]];`



Array methods (2)

- Arrays are dynamic. New elements can be added using assignment statements
- Example:

```
var a = [1,2,3]; // length is 3
a[3] = 5; // length is now 4
a[10] = 12; // length is now 11
```
- In last example `a[4]` to `a[9]` are undefined



Array Methods (3)

- For an array `a` in Java the length of `a` is a read only value given by `a.length`
- In JavaScript it is a read / write property:
 - `var a = [1,2,3];`
 - `a.length = 100;`
 - `a[99] = 123;`
- Now `a` has length 100 and the elements `a[3]` to `a[98]` are undefined.



Array Methods (4)

- The **join** method converts all array elements to strings and concatenates them using, by default, a comma to separate elements. The separator can be specified:
- Example:

```
var a = [1,2,3];
```

```
var s = a.join();    // gives "1,2,3"
```

```
var s = a.join(":"); // gives "1:2:3:"
```




Array Methods (5)

- The **reverse** method reverses the order of the elements

- Example:

```
var a = [1,2,3];
```

```
a.reverse(); // gives [3,2,1]
```



Array Methods (6)

- The **sort** method sorts the array in place. The default is to temporarily convert elements to strings and sort alphabetically.
- Example:

```
var a = [2,1,3];  
a.sort(); // gives [1,2,3]
```
- A comparison function can be supplied as an argument.



Array Methods (7)

- Sorting an array of integers in decreasing order

```
var a = [1,2,3,4];
```

```
var decrease =
```

```
    function(a,b) { return b-a; };
```

```
a.sort(decrease);
```

- Now a is [4,3,2,1]



Array Methods (8)

- Sorting an array of strings in decreasing order

```
var a = ["one", "two", "three", "four"];  
var decrease =  
    function(a,b)  
    { return b.localeCompare(a);  
    }  
a.sort(decrease);
```

- Now a is ["two", "three", "one", "four"]



Array Methods (9)

- The `concat` method concatenates elements to the end of an array.
- Example:

```
var a = [1,2];  
var b = a.concat(3,4); // gives [1,2,3,4]
```
- Note that `concat` does not change the array `a`. It creates a new one.



Array Methods (10)

- The **slice** method returns a new array that is a subarray of the array.
- Example:

```
var a = [10,11,12,13,14,15];  
var b = a.slice(2,5); // returns [12,13,14]  
var c = a.slice(2); // returns [12,13,14,15]
```
- Note: **slice** does not change the array **a**. It creates a new one.
- Note: second index is one past last one used



Array Methods (11)

- The **splice** method can insert and/or remove elements from anywhere in an array. It modifies the array and also returns a new one.
- It's a classic example of a badly defined method that does too many things.



Array Methods (12)

- The **push** and **pop** methods treat the array as a stack with the top of the stack at the end of the array
 - `var s = [] ;` empty stack
 - `s.push(1) ;` s is [1], returns length 1
 - `s.push(2,3) ;` s is [1,2,3], returns length 3
 - `var top = s.pop() ;` s is [1,2], returns 3



Array Methods (13)

- The **unshift** and **shift** methods treat the array as a stack with the top of the stack at the start of the array
 - **var** s = []; empty stack
 - a.**unshift**(1); s is [1], returns length 1
 - s.**unshift**(2,3); s is [3,2,1], returns length 3
 - **var** top = s.**shift**(); s is [2,1], returns 3



Array Methods (14)

- Example document illustrating array methods:

<examples/simple/arrayMethods.html>



String methods (1)

- Create a new string object
 - `var s = "Hello";`
 - `var s = new String("Hello");`
- Length property of a string (read only)
 - `var len = s.length;`
- return a character of a string: There is no char type in Javascript: a char is a one-char string
 - `var c = s.charAt(n);`



String methods (2)

- Return character code (Unicode value)
 - `var code = s.charCodeAt(n);`
- Concatenate one or more strings `s1, s2, ...`
 - `var s = s1.concat(s2, ...);`
 - `var s = s1 + s2 + ...;`
- Create string from Unicode values `c1, c2, ...`
 - `var s =
String.fromCharCode(c1, c2, ...);`



String methods (3)

- Find first index of **pattern** in a string **s**
 - `var index = s.indexOf(pattern);`
 - `var index = s.indexOf(pattern, startIndex);`
- Find last index of **pattern** in a string **s**
 - `var index = s.lastIndexOf(pattern);`
 - `var index = s.lastIndexOf(pattern, startIndex);`



String methods (4)

- Compare strings in a locale dependent way.
 - `var result =`
 `s1.localeCompare(s2) ;`
- `result` is less than zero if `s1` precedes `s2`
- `result` is zero if `s1` equals `s2`
- `result` is greater than 0 if `s1` follows `s2`



String methods (5)

- Matching, replacing, or searching a string `s` for a pattern specified by a regular expression object `regex`.
 - `var a = s.match(regex);`
 - `var s = s.replace(regex, replacementString);`
 - `var index = s.search(regex);`
- We will consider regular expressions later



String methods (6)

- Create a new string from a slice of a string `s`. String `s` is not modified.
 - `String s1 = s.slice(start, end) ;`
- `start` is the first index of the slice and `end-1` is the last index



String methods (7)

- Split a string `s` into an array of strings using a `delimiter` string or regular expression to specify the split points
 - `var a = s.split(delimiter);`
 - `var a = s.split(delimiter, maxLength);`
- The optional `maxLength` argument specifies a maximum size for the array `a`.



String methods (8)

- Create a string that is a substring of string `s`
 - `var sub = s.substring(start, end);`
- The substring begins at index `start` and ends at index `end - 1`;
- Note: There is also a `substr` method which is deprecated.



String methods (9)

- Create upper or lower case versions of string `s`
 - `var s1 = s.toLocaleLowerCase();`
 - `var s1 = s.toLocaleUpperCase();`
 - `var s1 = s.toLowerCase();`
 - `var s1 = s.toUpperCase();`
- Note: Strings are immutable so string `s` is not modified by any of these operations.



Global object (1)

- This object has properties and methods that don't fit anywhere else
- Properties
 - `Infinity`
 - `NaN`
 - `undefined`



Global objects (2)

■ Methods:

- `decodeURI(uri)` ,
 `decodeURIComponent(s)`
- `encodeURI(uri)` ,
 `encodeURIComponent(s)`
- `escape(s)` , `unescape(s)`
- `eval(s)`
- `isFinite(n)` , `isNaN(x)`
- `parseFloat(s)` , `parseInt(s)`



Global objects (3)

- Convert a string `s` to an integer or floating point number
 - `var n = parseInt(s) ;`
 - `var n = parseFloat(s) ;`
- Note: these functions return the first number found at the beginning of string `s`. If `s` does not begin with a number `NaN` is returned and can be tested using the `isNaN(n)` function.



Global functions (4)

- Encoding and decoding a string *s*
 - `var e = escape(s) ;`
 - `var f = unescape(s) ;`
- `escape` returns an encoded version of *s* in which special characters are represented in the form `%xx` or `%uxxxx` (Unicode) where *x* is a hex digit: Example
 - `escape("Hello World")` is `"Hello%20World"`
- `unescape` decodes an encoded string



Global functions (5)

- JavaScript can be constructed and executed at run time using the **eval** function:
 - **eval**(expression) ;
- Here expression is any string that contains JavaScript code
 - **var** expr =
 "Math.sqrt(x*x + y*y) ;";
 - **var** h = **eval**(expr) ;



Math object (1)

- `Math.abs(x)`
- `Math.acos(x)` , `Math.asin(x)` ,
`Math.atan(x)` , `Math.atan2(y,x)`
- `Math.ceil(x)` , `Math.floor(x)`
- `Math.sin(x)` , `Math.cos(x)` ,
`Math.tan(x)`
- `Math.E` , `Math.PI` , `Math.LN10` ,
`Math.LN2` , `Math.LOG10E` , `Math.LOG2E`
- `Math.exp(x)` , `Math.log(x)`



Math object (2)

- **Math.max**(v1, v2, . . .)
- **Math.min**(v1, v2, . . .)
- **Math.pow**(x, y)
- **Math.random**()
- **Math.round**(x)
- **Math.sqrt**(x)



Date class (1)

- The current date and time:
 - `var today = new Date();`
- General constructor for day (1 to 31), month (0 to 11), year (4 digits)
 - `var d = new Date(year, month, day);`
- More general form:
 - `var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);`



Date class (2)

- There are at least 30 methods in this class:
 - `getDate()` , `getUTCDate()`
 - `getDay()` , `getUTCDay()`
 - `getFullYear()` , `getUTCFullYear()`
 - `getHours()` , `getUTCHours()`
 - `getMilliseconds()` , `getUTCMilliseconds()`
 - `getMinutes()` , `getUTCMinutes()`
 - `getMonth()` , `getUTCMonth()`
 - `getSeconds()` , `getUTCSeconds()`
 - `getTimezoneOffset()`



Date class (3)

- There are also the corresponding set methods
 - `setDate(...)`, `setUTCDate(...)`
 - `setFullYear(...)`, `setUTCFullYear(...)`
 - `setHours(...)`, `setUTCHours(...)`
 - `setMilliseconds(...)`,
`setUTCMilliseconds(...)`
 - `setMinutes(...)`, `setUTCMinutes(...)`
 - `setMonth(...)`, `setUTCMonth(...)`
 - `setSeconds(...)`, `setUTCSeconds(...)`
 - `Date.parse(...)`, `Date.UTC(...)`



Date class (4)

- Converting to strings
 - `toDateString()` , `toUTCString()`
 - `toLocaleDateString()`
 - `toLocaleString()`
 - `toLocaleTimeString()`
 - `toString()` , `toUTCString()`
 - `toTimeString()`

<examples/simple/date.html>



The RegExp object

- It represents regular expressions
- More on this later



Custom Objects

- JavaScript is a prototype based OOP language rather than a class based one.
- Objects can be directly defined
 - `var point = { x:1, y:2 };`
 - Now `point.x` is 1 and `point.y` is 2
 - `var circle = {x:1, y:2, radius:3};`
 - Now `circle.x` is 1, `circle.y` is 2 and `circle.radius` is 3
- This is like struct in C

Point objects (1)

```
function Point(x,y)
{
    this.x = x;
    this.y = y;
}
```

two instance
methods

```
Point.prototype.toString = pointToString;
Point.prototype.distance = pointDistance;
Point.distance = point2Distance2;
```

class method

- Except for **this** it's like an ordinary function
- Now we need to define the instance methods



Point objects (2)

```
function pointToString(x,y)
{
    return "(" + this.x + "," + this.y + ")";
}
function pointDistance()
{
    return
        Math.sqrt(this.x*this.x + this.y*this.y);
}
```



Point objects (3)

```
function pointDistance2 (p1,p2)
{
    dx2 = (p2.x - p1.x) * (p2.x - p1.x) ;
    dy2 = (p2.y - p1.y) * (p2.y - p1.y) ;
    return Math.sqrt(dx2 + dy2) ;
}
```



A static method

Point objects (4)

```
<script>
```

```
var p = new Point(1,2);
```

```
document.write("p = " + p);
```

```
document.write("<br>x = " + p.x);
```

```
document.write("<br>y = " + p.y);
```

```
document.write("<br>Distance from origin is " +  
    p.distance());
```

```
</script>
```

toString is used
automatically
as in Java

data fields (properties)
are always public



Point objects (5)

```
<script>
```

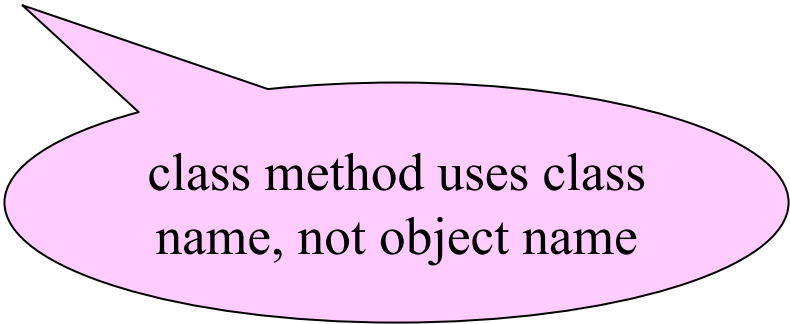
```
var p1 = new Point(1,2);
```

```
var p2 = new Point(3,4);
```

```
var d = Point.distance(p1,p2);
```

```
document.write("<br>", "Distance from " + p1  
    + " to " + p2 + " is " + d);
```

```
</script>
```



class method uses class
name, not object name



Circle objects (1)

```
function Circle(p, r)
{
    this.center = p;
    this.radius = r;
}
```

```
Circle.prototype.toString = circleToString;
```

```
Circle.prototype.circumference =
    circleCircumference;
```

```
Circle.prototype.area = circleArea;
```

- Except for `this` it's like an ordinary function
- Now we define the instance methods



Circle objects (2)

```
function circleToString(x,y)
{
    return this.center + ":" + this.radius;
}
function circleCircumference()
{
    return 2.0 * Math.PI * this.radius;
}
function circleArea()
{
    return Math.PI * this.radius * this.radius;
}
```



Circle objects (3)

```
<script>
var p = new Point(1,2);
var c = new Circle(p,3);
document.write("c = " + p);
document.write("<br>", "Center is " + c.center);
document.write("<br>", "Radius is " + c.radius);
document.write("<br>", "Circumference is " +
    c.circumference());
document.write("<br>", "Area is " + c.area());
</script>
```

<examples/simple/objects.html>



More on Objects

- There is much more to objects in JavaScript
 - prototypes
 - existing objects can have new properties added to them (even built-in objects)
 - Can have both instance and class properties or methods
 - prototype based inheritance



Associative arrays

- Object properties can be referenced directly
`circle.center`
or they can be accessed as elements of
associative arrays as in
`circle["center"]`
using the property name as a string